Software at g-2

Adam Lyon (FNAL/CD)

21 June 2011

Note:

Nothing here is set in stone. Feel free to critique, complain, make suggestions (and offer to help).

Why do we need a software system?

<1> Science demands reproducibility.
We must have control over our software.

<2> We want to work together.
Sharing code is a good thing.

<3> We want to do physics, not computing.
Don't want every physicist to have to deal with low level code details

Three pieces

<A> Source code revision control
Be able to reproduce our code. Keep track of changes

<B> Build system
Easily build code without losing all your hair

<C> Release/environment system
Be able to run standard code and use standard libraries from the above. Setup a standard environment

All of these pieces interact with each other

We don't want to be making wheels

"Art" is a common framework based on CMS being worked on by the FNAL/CD/CET group. They are committed to long term support. Adopted by NOvA, Mu2E, LBNE*

Desire to move in this direction. Since we don't have much code yet, we are in a great position do to this!

The Art folks have much expertise in

- Redmine/Git
- cmake
- Relocatable ups

I want to go down this road too. They can help us.

<A> Source code revision control

Redmine! It's awesome.
http://cdcvs.fnal.gov/redmine/projects/g-2
Can view the repository from Redmine

git - it's a younger, more spry CVS/SVN
- Used by most every important open source project
- Designed for easy branching, merging, sharing
- Designing for coding off the grid

E.g.
There is now a "g2migtrace" project in Redmine using a git repository. Please do not use the "gm2" project any longer.

Will talk about this today!

<B> Building code

The art folks use "cmake" - a system for generating and managing makefiles.

Pretty easy to understand (but hopefully you will just "use" it)

Integrates easily with g2migtrace's build system

Eventually - use "BuildBot" for automated builds.

<C> Release / environment control

You don't want to build *everything* yourself! Need a "repository" of executables and libraries with their associated runtime files

Relocatable ups **(the dinosaur grows wings!)**

"setup gm2 v2_4"   sets up your environment

Set your PATH for executables
Sets your LD_LIBRARY_PATH for libraries
Sets other necessary environment variable for finding runtime files, headers

<C>

Relocatable ups - can hold multiple versions of products. Can handle product dependencies

Can install by only unwinding tar files!!

Easy for remote sites.

Can "layer" UPS product areas...

External UPS (Art, Geant, root)
Comes from tar files from FNAL/CD/CET

g-2 UPS (our code)

Your local UPS

Your development area

Where are we right now?

g2migtrace was migrated to git! New Redmine project at https://cdcvs.fnal.gov/redmine/project/g2migtrace

I have some scripts to setup a development area and get code from Redmine

Building g2migtrace with "cmake" works (see the tutorial)

I have the beginnings of a ups area for g-2. Art, root, geant4, CLHEP, Boost, … are installed

What you can do?

Please try the tutorial and let me know what you think

Expect things to change (at least a little) as more of this work gets figured out

Please use the git version of g2migtrace (I'll eventually "archive" the SVN version).

--- GIT ---

If you've used CVS or SVN, git requires thinking a little differently (but you'll like it)

CVS/SVN:
There is a master repository. You checkout code to your area. You make changes. Check code back in to the master repository.

If you are brave/crazy, you work on a branch (check in changes that don't affect the mainline development).

You must be on the network to do most repository commands

With git...

You "clone" a REPOSITORY from a central place (Redmine in our case)

You check in code on YOUR repository. This is all LOCAL to you! No one else sees this.

To Share:
You "push" your commits back to the central place (Redmine)
You "pull" commits from others from the central place

You're repository is isolated until you push/pull. You can commit code to change all muons to croutons and it's ok. No one will know until you push the changes.

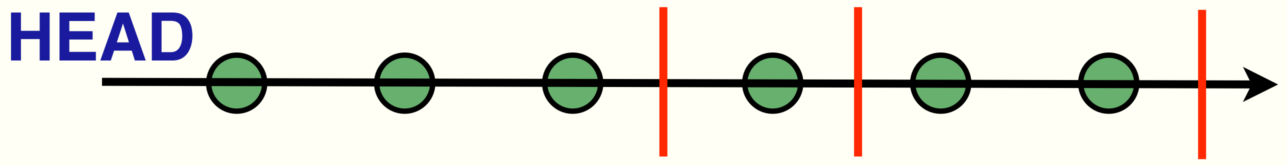
COMMIT EARLY - COMMIT OFTEN - PUSH WHEN READY TO SHARE

With git...

Branching/merging with CVS is insane

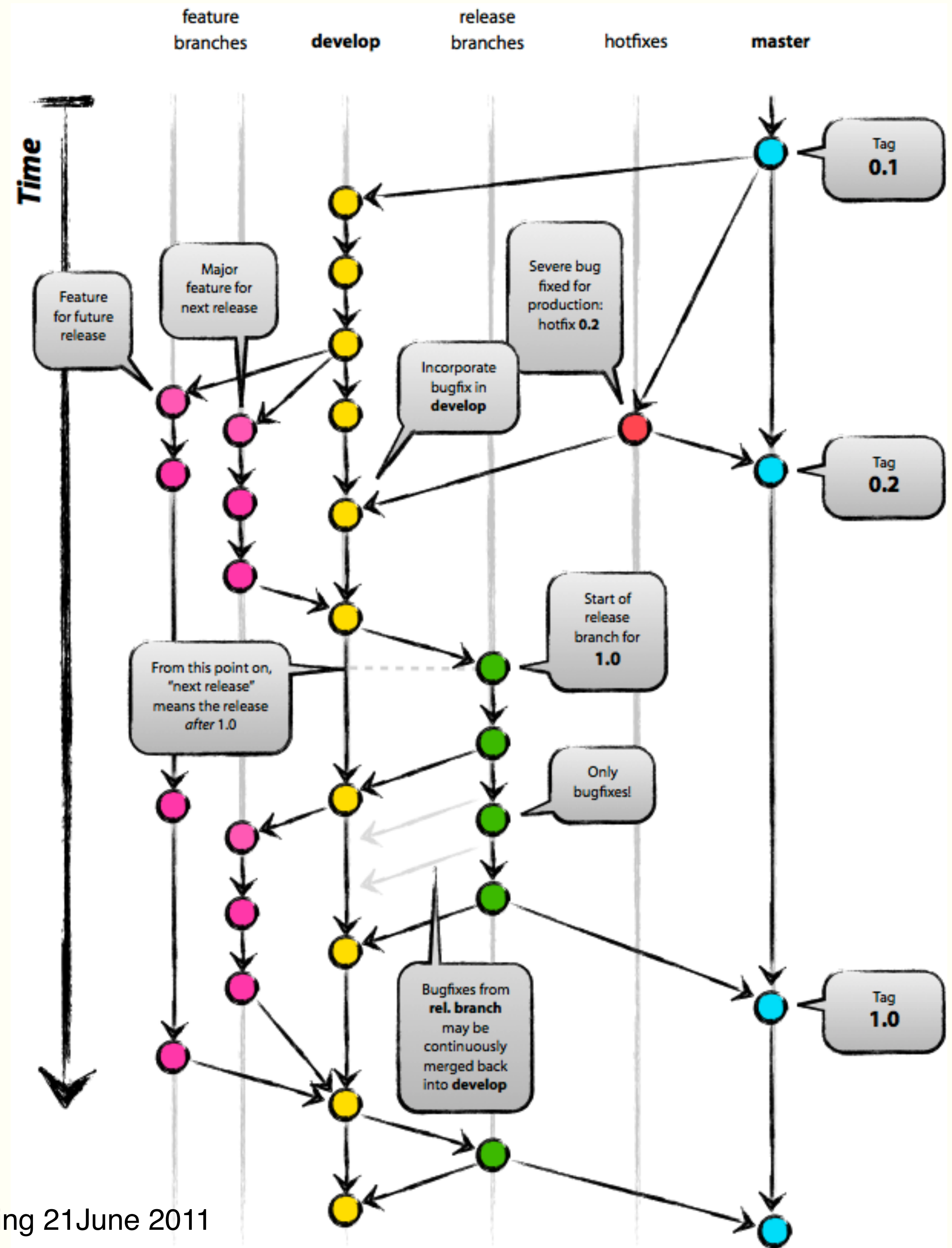
Branching/merging with SVN is better, but still hard

Branching/merging with Git is easy and fun!



Common use of CVS HEAD. It always has to work. Hard to make bug fixes on older code while doing development. Hard to share code

With git…


feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Feature for future release
Major feature for next release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

You can effectively use branches to make your life easier.

Easy to commit changes without affecting others

Easy to keep a branch clean (like master)

Easy to make hotfixes

With git… Tracking branches

Your commits and branches are LOCAL TO YOU unless you push/pull.

Git remembers the "origin" of the clone.

You can setup a "tracking" branch. Branch is "linked" to a branch on Redmine - you can backup your code and share your code [NOTE: YOUR COMMITS ARE STILL LOCAL UNTIL YOU PUSH]

git push/pull automatically remembers your branch is a tracking branch and will do the right thing.

This was complicated for me at first -- but ...

Problems with git

It's not trivial.

Many git commands; many options

Certain operations require several commands. Must issue in the right order.

To make things easy use "git flow". A wrapper around git for easy development workflow.

Things to get used to - two step committing

You are used to "cvs ci -m 'My message' myFile.cpp"

In Git, you "stage" commits.

"git status" to see what has changed

"git add <file>" to 'stage' files for committing

"git commit -m <message>" to commit

OR can do in one shot with
git commit -m <message> file1 file2 file3

But I actually like using the staging

Git commands

git status -- see what has changed
git diff <file>   -- what has changed for this file
git diff -- what has changed for everything
git log -- see the log

git add <file> -- stage a file/directory
git rm <file> -- delete a file from the repository
git mv <file> -- move/rename a file within the repository
git commit -m <message> -- commit

git branch -- see what branch you're on
git checkout <branch> -- switch to a different branch

For tracking branches
git push -- put your commits onto redmine to branch
git pull -- pull commits from others from redmine to your branch

Git flow commands

Feature branches

git flow feature start <newFeature>

git flow feature finish -- merges with the develop branch

git flow feature publish -- creates branch on redmine & tracks

git flow feature track -- Prepare to track a redmine feature branch

Also git flow hotfix…, git flow release…

Makes branching/merging really easy! Don't have to remember lots of git trivia.

I have some commands for setting up git repositories.

gm2d newDev   -- make a new development area

gm2d getRedmineGit <name>   -- clone a Redmine repository

There's a LOT more to git (do physics instead)

Good references…

Git community book: http://book.git-scm.com
Pro Git: http://progit.org

Git flow:
http://yakiloo.com/getting-started-git-flow
(Note that gm2d getRedmineGit does git flow init for you)
[And makes develop a tracking branch]

Now on to the tutorial

This tutorial:
How to make a development area, checkout g2migtrace, build it, do some development, push those changes back, share code on a feature branch -- should be enough for you to start development with g2migtrace!!

Future tutorials/documentation:

1) How to make a new Redmine project and develop
2) Eeek - I've messed up my Git repository and my muons are croutons! What do I do now?

Future things to do:
Buildbot
Make release scripts

Remember the NOTE from the beginning